

REMARKS

Claim 1 is amended to call for "a register" on line 11 of Claim 1. This is to overcome the examiner's rejection under 35 U.S.C. 112, second paragraph.

Claims 1-6 are rejected under 35 U.S.C. 103(a) as being unpatentable over Rau et al "Code Generation Schema for Modulo Scheduled Loops", 1992, pp 158-169 (hereinafter Rau_1) in view of Rau et al, "Register Allocation for Software Pipelines Loops", June 1992, Proceedings Of the ACM SIGPLAN'92 Conference on Programming Language Design and Implementation, pages 283-299 (hereinafter Rau_2), and further in view of Akkary, USPN 6,240,509 (hereinafter Akkary).

Applicant's claim 1 call for: "A method of pipelining program loops having irregular loop control comprises the steps of:

determining which instructions in loop code in a memory may be speculatively executed without hardware support, special loop control instruction ,

storing in a computer memory a set of registers that are modified by an instruction and are alive out of the loop, and

modifying the program code so that the values of those registers are saved to a temporary register during all proper iterations, and

copying back to a register the value of the temporary register once the loop is completed."

In Rau_1, the authors do claim that an irregular loop can be software pipelined if the entire epilop can be collapsed. Obvious or not, this is not the contribution of applicants' invention.

The primary challenge of collapsing the epilog for an irregular loop is two fold: recurrence constraints and register pressure (esp. predicate pressure). Recurrence constraints (aka recurrence cycle, dependence cycle) refer to the order in which operations must execute. In processing the irregular loop so that the epilog can be collapsed, it is a challenge to minimize the number of new instructions added on the critical cycle. Lengthening this critical cycle slows down the loop by slowing down the rate at which new iterations can be initiated.

The contribution of applicant's invention is software pipelining irregular loops *without* the special purpose hardware and with a limited NON-rotating register file, and a very small predicate register file (5 or 6 predicate registers). Secondly, there are a limited number of slots and we MUST avoid drastic increases in code size (e.g., multiple epilogs).

Applicants' teach the addition of register copying which is a novel process of inserting code into the loop, when necessary, to save the state of live-out registers (i.e., registers which may be read after the loop before being written) which are modified by non-speculated instances of instructions. The results are copied to temporary registers after it has been determined that the instruction would have been executed in the original stream. After loop execution, the original registers are restored with the last value written to the temporary register. Register copying is preferred over predication on architectures without special hardware support because the WHILE loop can be pipelined more efficiently.

It is NOT obvious that this can be done effectively in practice with a small register file and predicate register file. In fact, the first version of applicants' paper was rejected because the paper reviewers were skeptical that applicants' method would work.

The minimum hardware that Rau_1 relies on is hardware for speculative execution. This is a non-significant, complex piece of hardware that is not realistic for all processors, especially many in the embedded world for a variety of reasons, including cost and power. These are not issues in the Rau world (general purpose processors). Their alternate hardware configuration has rotating register and hardware for speculative code motion.

Rau_2 depends on dynamic registers (aka rotating register files) or MVE (software-only approach) to software pipeline irregular loops. MVE has a big drawback: code size! With MVE, loop unrolling (making multiple copies of a loop body and multiple epilogs) are **required**, both are **very** expensive in terms of code size. Additionally, loop unrolling often increases register pressure significantly (from our experience - mileage may vary) which is problematic for loops that are already register-pressure bound.

The Akkary reference relies on trace buffers for speculative execution, again a non-trivial piece of hardware. Again, it's a different ballgame when you can put this type of hardware into your processor.

It is not seen where this combination with the conditional copying back step is taught or suggested in the references. Neither Rau_1, Rau_2 nor Akkary teach modifying the program code so that the values of those registers are saved to a temporary register

during all proper iterations and then copying back to a register the value of the temporary register once the loop is completed.

Furthermore, in regard to combining the cited prior art, reference is made to *In re Fritch*, 23 USPQ2d 1780 and particularly the portion thereof at page 1783 under "Prima Facie Obviousness" where the Court stated:

"In proceedings before the Patent and Trademark Office, the Examiner bears the burden of establishing a prima facie case of obviousness based upon the prior art. '[The Examiner] can satisfy this burden only by showing some objective teaching in the prior art or that knowledge generally available to one of ordinary skill in the art would lead that individual to combine the relevant teachings of the references.' The patent applicant may then attack the Examiner's prima facie determination as improperly made out, or the applicant may present objective evidence tending to support a conclusion of nonobviousness."

and later stated:

"Obviousness cannot be established by combining the teachings of the prior art to produce the claimed invention, absent some teaching or suggestion supporting the combination. Under section 103, teachings of references can be combined only if there is some suggestion or incentive to do so.' Although couched in terms of combining teachings found in the prior art, the same inquiry must be carried out in the context of a purported obvious 'modification' of the prior art. The mere fact that the prior art may be modified in the manner suggested by the Examiner does not make the modification obvious unless the prior art suggested the desirability of the modification."

There is no suggestion in the references to suggest the combination claimed or the desirability of the modification.

Claim 1 is therefore deemed allowable.

Claims 2 and 3 dependent on claim 1 are deemed allowable for at least the same reasons claim 1 is deemed allowable.

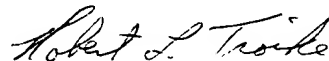
Claim 4 calls for "A method for software pipelining of irregular conditional control loops includes pre-processing the loops so they can be safely software pipelined, comprising the steps of:

pre-processing each instruction in the loop in turn;
if the instruction can be safely speculatively executed, leaving the instruction alone;
if it could be safely speculatively executed except that it modifies registers that are live out of the loop, pre-processing the instruction using register copying and otherwise using predication."

This is not taught in Rau_1 or Rau_2 or Akkary for the reasons discussed above. Claim 4 is therefore deemed allowable. There is no suggestion in either reference of register copying or "if it could be safely speculatively executed except that it modifies registers that are live out of the loop, pre-processing the instruction using register copying and otherwise using predication."

In view of the above applicants Claims 1-4 are deemed allowable and an early notice of allowance of these claims is deemed in order and is respectfully requested.

Respectfully requested;



Robert L. Troike (Reg. 24183)

Telephone No.(972) 917-4360